# **Advanced Pandas**

### **Take pandas to the next level**

Selecting transcript lines in this section will navigate to timestamp in the video

- Python without Pandas is like skiing without skis. You could do it but it'd be a heck of a lot harder. In this course, we're going to take Pandas from the basics to the next level. I'm Brett Vanderblock, Data Scientist at Patagonia and Pandas has been there every step of my journey to becoming a data scientist and producing insightful analytics. We'll be covering the most critical functions Pandas has to offer from transforming data frames to advanced calculations to techniques for rapid exploratory data analysis. We'll also take a peak beyond Pandas for geospatial and big data processing. I want you to get the most out of your data. Now let's dive in to Advanced Pandas.

### **Getting started with pandas**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Import pandas as pd. This is probably the single most common line of Python code you'll find in data science. And there's a reason for that. Pandas is the easiest and most intuitive Python package for dealing with your data. With it, you can transform, clean, visualize, and extract insights from your data with relative ease. Throughout this course, we'll explore the most fundamental aspects of pandas to make your work flow more easily and efficiently. Then, we'll expand to more advanced applications of pandas and showcase a few other packages related to pandas that can take your work to the next level. Depending on your Python distribution, you may already have the pandas package installed. If not, there's two options to install the package. You can PIP install pandas via command line, or if using the Conda installer, Conda install pandas. This course consists of Jupyter Notebooks like the one you see here and requires a Python installation. If you don't have Jupyter or Python, I recommend you download the distribution from Anaconda. The link for this download is available here.

### **Intro to DataFrames using pandas**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] The most fundamental aspect of Pandas is the DataFrame. This is how your data is stored, and it's a tabular format with rows and columns as you'd find in a spreadsheet or a database table. So before we dive into some more advanced Pandas topics, let's review the DataFrame concept. After importing Pandas as PD, we're going to create a dictionary called scores. Now a dictionary is a Python structure which stores key value pairs. In this dictionary, the keys are name, city, and score, and the values are lists, as denoted by the square brackets which are mapped to their corresponding key. After running this cell, we're going to turn this dictionary into a Pandas DataFrame using the DataFrame function. Note the capitalization of the F in DataFrame. Great. Now let's see our data. Here you can see a table with name, city, and score as column headers, and three rows of corresponding data. Each column is a series. And notice the values zero, one, and two to the left. These are the index of our DataFrame and are useful for referencing and subsetting our data. If we wanted to just return one column in our DataFrame, the notation is your DataFrame and then the column name or names in square brackets. Here let's take a look at score. Great. Note, in this example, you can also call DF.score to return the same result. Similarly, you can also create new columns in your DataFrame by passing a new column name into the square brackets and assigning it. Here, we're creating a new column that combines the name and city columns. Now let's say we wanted to subset our data to only show those folks with scores above, say 90. To do that, we can create a Boolean expression which returns true for scores greater than 90 and only return those records where this condition is true. After running, we returned a DataFrame with just Japhy and Zosa's records. Also note our new column, name\_city. All this only scratches the surface of what you can do when your data is in a DataFrame, but this is an excellent start for us to build on for future lessons.

### **Top functions using pandas**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] As you use pandas, you'll find there are certain functions that prove their worth time and time again. In this lesson, we'll cover some of the most important functions that you can use to get more from your data. Pandas is very flexible in that you can import data from a wide variety of data sources, including CSVs, Excel files, JSONs, databases, parquet files, you name it. For this lesson, we'll use panda's read CSV function to import the iris dataset as a data frame. This is a common sample dataset for practicing data science. Import pandas as pd. Next we'll read the iris CSV into a data frame called iris. Data frames have an attribute called shape which tells us the dimensionality of our data. By calling iris.shape, we can see the number of rows and columns that our data frame has. There you go. We have 150 rows and five columns in our data. To preview our data, the head function will return the top rows of our data frame. Let's check out the top three rows of the iris data. You can see this data contains measurement data for different iris species. Similarly, you can also view the bottom rows with the tail function. Now, when working with data in pandas, you'll find that the data types that pandas assigns to your data is important and will influence what operations you can perform. I have found pandas to be pretty intelligent in how it assigns data types, but you'll want to check to be sure. To do this, call the dtypes attribute on your data frames. We see two data types represented in our data frame; float for all the measurement data, and object for the species. Often when using pandas, you'll want to subset your data. Loc allows you to subset your data based on index labels, so either the row indexes or column names. Iloc subsets by position, so the row number or column order. So here, we're going to subset our data frame based on row indexes three, four, and five, which are the fourth through sixth rows of our data. Note indexing begins at zero. We can also return a single cell value by passing a row and column names separated by a comma. This returns 4.6, which is the measurement for sepal length for the row at index three in our data frame. Using iloc, we can return the same value by referencing the same row index of three but a column index of zero. Often, after you've done a whole host of data transformation with pandas, you want to export your data frame for analysis or visualization. A handy way to do this is the to\_csv function. Note you may want to include index equal to false so the index isn't included in your CSV. Great. This will have generated a CSV named iris-output in your working directory. These functions are so beneficial in data analysis, and if you aren't using them currently, I highly recommend you give them a try.

### **Configuring options using pandas**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] So another Pandas feature I want you to take advantage of, is to configure your own options. Pandas has an option system which allows you to customize how the package works for you. Most often, this can be useful to change how results are displayed in Pandas. So here's an example. We'll start with this sample data frame of the top three nations in 2018 by global carbon dioxide emissions. The first option which comes in handy is to configure the maximum row size to display for a Pandas data frame. If we set the max row size to two, here's what we get. So you see two rows displayed, separated by an ellipses. That's what this option does. You can either use it to limit the screen space your displayed data frames take up or conversely to expand the row size, to reveal more of your data. Similarly, the max columns display option will reveal or hide columns. I find this most useful when viewing the head of a data frame that has a lot of columns as Pandas will truncate these by default. Now we see another set of ellipses between our first and third columns. Another trick worth checking out is to suppress scientific notation for displaying floats. You'll notice the scientific notation makes it difficult to readily compare carbon dioxide emissions figures in our table. By modifying the float format option, you can display these values normally and even add in a comma as 1,000 separator. So there you have it. Now you've got one more trick up your sleeve by customizing your options in Pandas.

### **Data type conversions using pandas**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] One area where you might encounter some hurdles with pandas is dealing with data types. pandas is generally pretty good at assigning proper data types but, nonetheless, you'll find many instances when you need to convert data types. In this lesson, we'll leverage the planets dataset as it has a good variety of data types. After importing pandas, we'll read the planets data as a CSV. (typing) Now, let's have a peek at the data. From looking over the data frame, you can probably infer what the data type assignments will be, but to be sure we can access the dtypes attribute of planets. Now, we see data types varying from an object to integers to floats. How pandas handles your data depends on the data types you've designated. For example, we'll use the mean function to return the average for all float and integer values in our dataset. Everything looks pretty good, though you might question whether it really makes sense to take an average of a year, as we've done here. Let's see how different data types interact. Here, we'll divide an integer column by a float. The result is a float, great, that's what you'd hope for. We also have the ability to change data types using the astype function. For instance, we can convert the integer value of the number column to a float. It's useful to see what happens when you convert a float to an int. In this case, we've lost the decimal point and it's worth noting that this approach would effectively round down any floats as you convert to integers. We could also convert the year to an object by calling the astype(str) for string. To take advantage of the date time data type in pandas we can convert our integer year value to a date time using to\_datetime. Then, we specify how the data is currently formatted. So, you see here, our integer year has been converted to an actual date time corresponding to the first day of the year. I want you to unlock the full potential of pandas but before you can, there's a good chance you'll encounter some data type issues and now you'll be ready to solve them.

### **Data type conversions using pandas**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] One area where you might encounter some hurdles with pandas is dealing with data types. pandas is generally pretty good at assigning proper data types but, nonetheless, you'll find many instances when you need to convert data types. In this lesson, we'll leverage the planets dataset as it has a good variety of data types. After importing pandas, we'll read the planets data as a CSV. (typing) Now, let's have a peek at the data. From looking over the data frame, you can probably infer what the data type assignments will be, but to be sure we can access the dtypes attribute of planets. Now, we see data types varying from an object to integers to floats. How pandas handles your data depends on the data types you've designated. For example, we'll use the mean function to return the average for all float and integer values in our dataset. Everything looks pretty good, though you might question whether it really makes sense to take an average of a year, as we've done here. Let's see how different data types interact. Here, we'll divide an integer column by a float. The result is a float, great, that's what you'd hope for. We also have the ability to change data types using the astype function. For instance, we can convert the integer value of the number column to a float. It's useful to see what happens when you convert a float to an int. In this case, we've lost the decimal point and it's worth noting that this approach would effectively round down any floats as you convert to integers. We could also convert the year to an object by calling the astype(str) for string. To take advantage of the date time data type in pandas we can convert our integer year value to a date time using to\_datetime. Then, we specify how the data is currently formatted. So, you see here, our integer year has been converted to an actual date time corresponding to the first day of the year. I want you to unlock the full potential of pandas but before you can, there's a good chance you'll encounter some data type issues and now you'll be ready to solve them.

### **Working with strings using pandas**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Text data can be an incredibly rich source of data for analysis and Pandas is a well-equipped for working with, cleaning, and processing text data in string format. Let's dive into some useful methods. For this lesson, we'll start with a series of three names, organized last, first and you might notice some inconsistencies in how these names are stored. In Pandas, the string accessor denoted by .str enables a host of useful string transformations. Let's start by replacing that semi-colon. Great, you can see the replace function easily replaced the semi-colon with the comma. Now another string operator that can be useful is calling the length in the number of characters of each string in your series. Great, so we see the first two names have 14 characters a piece and the third name has 10 characters. Now, I noticed some trailing and leading spaces in the names, so this is a great opportunity to use strip to remove those spaces. We'll also return the length so we can see the difference. Looks like we trimmed a leading or trailing space off of each of the first two names. For consistency, we'll also go ahead and convert to entirely lower case using the .str.lower. Note, .str.upper acts exactly as you think it might. Next, I want to reverse the order of first and last name. Thankfully, we have a comma delimiting first and last, so we can use .str.split to separate these two. This creates what is known as a tupal of last name then first name for each of our names. This next trick is a clever method using list comprehension to swap the order of last name and first name for each name. Our last step is to again join first and last name, separated by a space. This gives us a list of names in order of first then last and its all been tidied up a bit. Now, when you encounter text data in your work, I encourage you to put these Panda string functions to work for you.

### **Working with dates using pandas**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Time series data is one of the most interesting and essential types of data that we work with. But dates can be tricky to deal with. Thankfully, Pandas has some excellent methods that we can put to use. To get started, we're going to generate a series of dates. The period\_range function in Pandas allows us to do just that. By specifying the starting date, followed by the frequency, and the number of periods, we return a series of dates which we'll pass into a data frame. Great. Now we've got a data frame of four dates starting January 1st, 2020. And each is separated by 30 days. One useful function when working with time series data is the diff function. Diff will calculate the difference from a prior period. And in this sense operates similar to a SQL lag function. Let's see the difference from the prior date in our data. Sure enough, our dates are 30 days apart. Now often you'll want to take a date and convert it to the first day of the month. Similar to using a date trunc function in SQL. One easy method is to access the values property of our date column. Then use astype datetime64 to convert to a date time. By passing M between the square brackets, our date is now at the first of the month. There we go. Now let's quickly check our data types. You'll notice our original date is actually a period data type. We'll go ahead and convert that to a datetime64 timestamp using the dt accessor and the to\_timestamp function. This will help with some further transformations we'll want to use. Now to subtract two dates, no special treatment is involved. Just subtract them. Here we'll see the number of days between our date and the first of the month. Makes sense. In a similar fashion we can even subtract our date difference from above, and get the expected outcome. Similarly, you can use the Timedelta function to specify a time span you want to add or subtract from your date. Lastly, when working with date data types, Pandas has several quick tricks accessible with the dt accessor. Here we'll return the actual day name corresponding to each of our dates with the dt.day\_name function. Very cool. Well, that's it for dates. Definitely check out the many other useful dt functions which are detailed in Pandas' thorough documentation online.

### **Dealing with missing data using pandas**

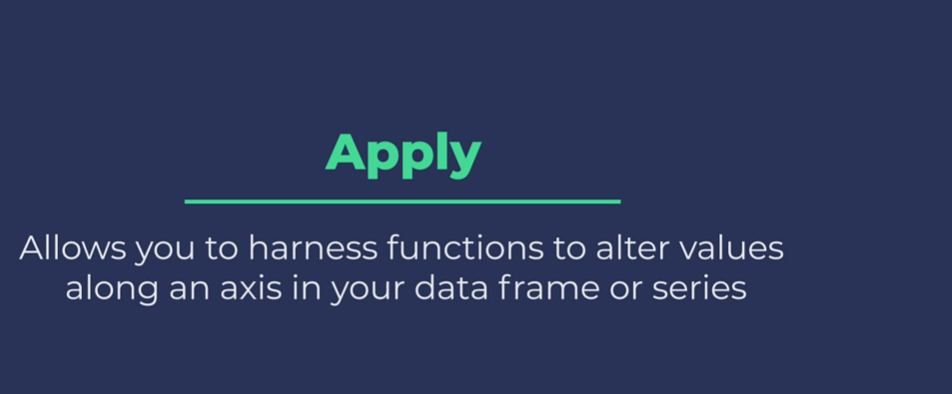
Selecting transcript lines in this section will navigate to timestamp in the video

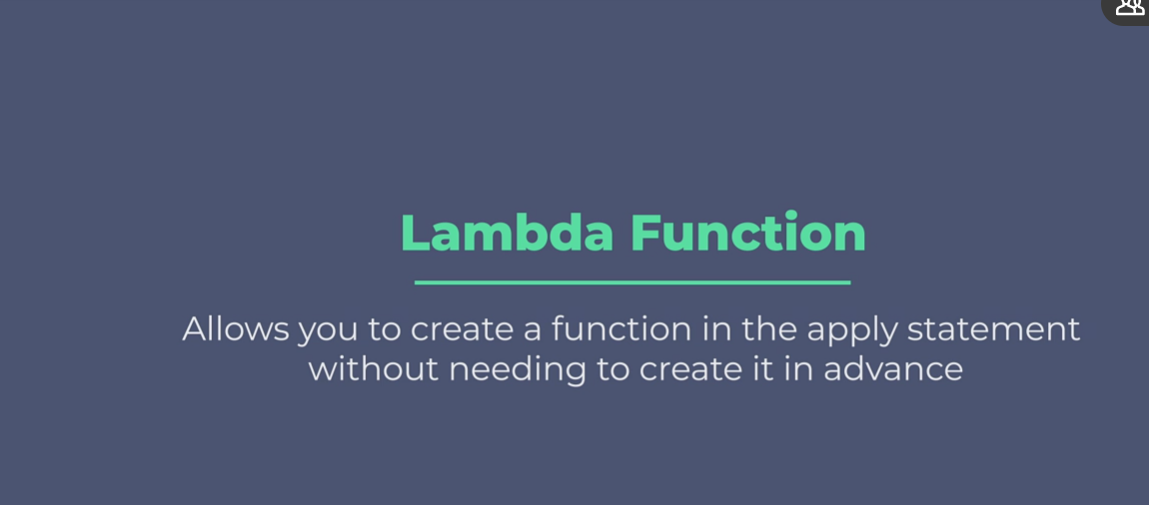
- [Instructor] When you first dive into your dataset, you may be surprised to find that some data simply isn't there at all. How you proceed to treat your data will have important ramifications down the line in your analysis. Let's review some approaches to dealing with missing data in Pandas. First, we'll create a data frame with temperature measurements. Here it is. Note the two missing values in sequence number four. One method to quickly identify all missing values in your data frame is to call isna. This will return true for any cells containing a missing value. Generally, the default parameters in Pandas functions are built to handle null values. For example, sometimes we'll treat nulls as zero and means ignore null values by default. Let's see an example using a cumulative sum down our data frame By default, the cumulative sum skips nulls. Now, if we set, skipna equal to false, the cumulative sum will null all subsequent results after the first null. One case where you'll need to be mindful of how Pandas treats nulls is when aggregating your data using group by. The default behavior is to exclude any records with no values for any dimensions you're grouping by. Let's see an example. Notice our entry with no measurement was not included. To prevent the group by from dropping nulls, pass dropna equal to false. Great, now let's review some methods to treat these nulls before you get too far along in your analysis. The most straightforward method is to simply drop records with null using dropna. Note, this method isn't without repercussions and you should consider this carefully before using this approach. By calling dropna, the default behavior is to drop any rows which contain null values in any column. Here, you can see that sequence four was dropped. Now, if you only want to drop rows with nulls in certain columns, you can use the subset parameter. A less common approach is to drop any columns with no values, which you can do by passing access equal to one in dropna. Now you can see we're left with just one remaining column without any nulls. Another method is to actually fill null values using fillna. To see this in action, we'll fill our nulls with zeros. At first glance, this could be problematic. Imagine if we were to calculate the mean for our temperature column. It would be heavily biased by the zero we just introduced. Another more nuanced approach is to use the pad method. This will carry over values from a prior row. Now this method poses its own issues, largely because we've simply created data out of thin air. Given the drop from 91 degrees to 50 degrees that we see, we might expect sequence four to fall somewhere in the middle. This brings us to our final method called interpolate. While interpolate allows for several different methods, the default approach will create a straight line estimate for our missing temperature value. There you go, now the estimate lies halfway between the two values. So before you get too far along analyzing your data, be sure to check for null values and put these methods to use.

### **Apply/Map/Applymap**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Python functions can be applied with great impact in pandas to alter data in your data frames. Thankfully, you don't have to create a for-loop to iterate through every row in your data to do this. In fact, it's not encouraged if you can help it. Pandas has frameworks that are simpler and more performance known as Apply, Map, and Applymap. Let's dig in. We'll start off with the data frame, including revenue and cost data for certain regions, teams, and squad. Now let's say you want to determine whether each squad was returning a profit or not. This means revenue exceeds their loss. This is an excellent application for Apply. Apply allows you to harness functions to alter values along an access in your data frame, or in your series. We'll save a bit more time by using a Lambda function. This allows you to create the function in the Apply statement without having had to create it in advance. Our Lambda function will return the string profit if revenue is greater than cost, otherwise it will return loss. The application of this Lambda function will return a series which we will use to populate the profit column in our data frame. After running, we see it worked as intended. In cases where revenue exceeds cost, we show profit and vice versa. Our next approach is called Map. And you can use it to alter values in a series only using a function dictionary or series. Here, we'll create a dictionary that maps teams to their corresponding color. Now we can map the values in the team column to a new column, which we'll call team color, looks great Another way to alter data in your data frame is to use Applymap. This applies a function to each element in your data frame. To show an example, we'll create a simple Lambda function which returns the character length of each value in our data frame. Now of course, there may be times where it's simply conceptually easier to formulate your logic as a for loop rather than one of the above. And that's okay. Let's see an example on action. Below, we're going to calculate each squads revenue as a percent of the region's overall revenue. We'll start with an empty list that we populate as we iterate through each row in our data frame. To construct the for loop, we loop through each eye in the range from zero up to the length of our data frame. Rev represents the revenue value for that particular row divided by the sum of all revenue in our data frame where the region equals this particular squads region. Lastly, we append this value before continuing the loop. Next, we set a new column revenue share of region equal to this list. Let's check out the result. The output looks great and the revenue share within each region sums up to a hundred percent. The methods we just implemented are among the most powerful in pandas. Between them, there's enough flexibility for just about any application you'll come across. Remember, Apply can be used for both data frames and series. Map works for series only. Applymap affects each element in a data frame. And when in doubt, for a loop should do the trick.

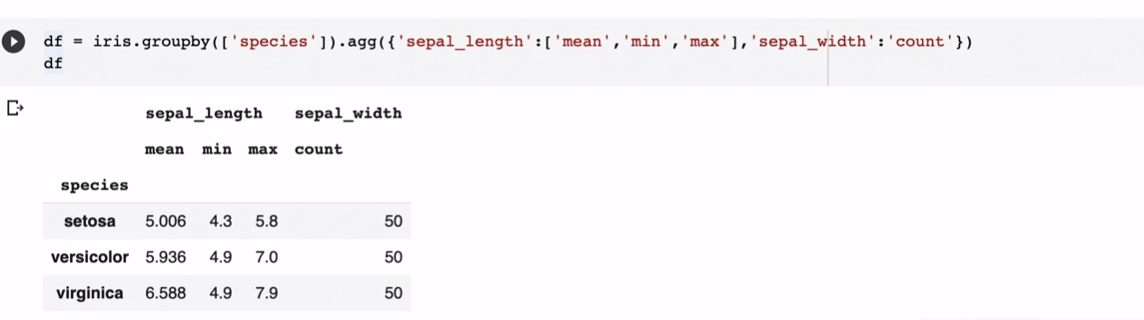


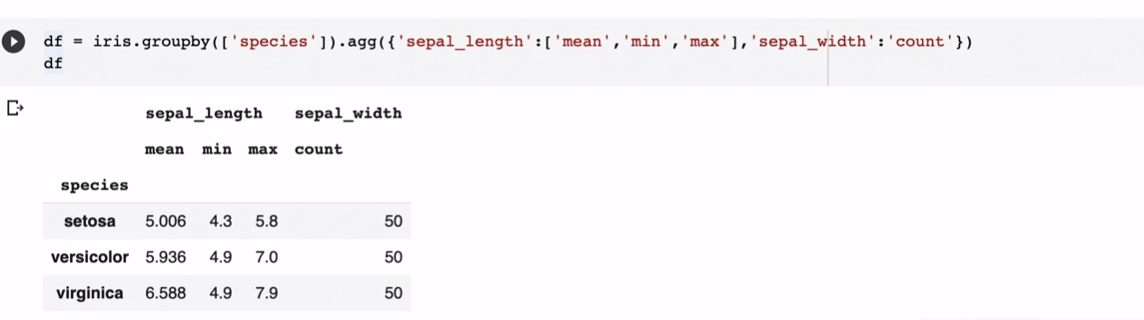


### **Groupby and aggregations using pandas**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Arguably the most fundamental transformation you can perform in Pandas is the group by function. Time and time again, this function proves its worth. So let's dive in. We'll start with the Iris data. Now let's apply a group by to Iris. In a group by, you determine the dimensions you want to group by, then specify an aggregation method. As a simple example, let's group by species and apply the maximum aggregation. The result gives us the max for each measurement. And we see we have three total species in our dataset. With group by, we can pass several different types of aggregations to multiple variables. To do this, call .ag after the group by and pass a dictionary with the variables you're interested in along with their associated aggregations. So for the sepal length, we return the mean, min, and max, and we'll count sepal width. Now here's our output. Notice the hierarchical column index that's created as a result. It's useful to see how this type of column index works. Let's select the sepal length column from our data. The result that's returned is all three aggregation types we chose for sepal length. Often, you'll want to flatten hierarchical indexes for simplicity. This approach uses join to concatenate the top and bottom level of our column index. The result combines our variable name with the aggregation used. You can also specify groupings without first applying any aggregation at all. Here, we'll create a grouping of Iris by species called groupings. Now with the groupings configured, you can reference the groupings for aggregating and filtering your data. The get group function operates like a where clause in SQL, filtering our data that is a part of the group you specify. Let's return the setosa group. Great, this yields all rows in our data with species of setosa. After defining your groups, you can later call aggregate functions. Here, we'll call max, and this returns an identical output to our max aggregation above. You can also apply Lambda functions to your groupings. For simplicity, we'll apply the max aggregation, which again, is identical to the output above. With groupings configured, you can use what is called filter to filter your data frame on an aggregate constraint. This is quite similar to a having clause you find in SQL. In our example, we're going to filter our data frame for only those species that have a maximum petal length below five. Looking at the table above, we can expect this will only return setosa species. The result is all 50 measurements from the setosa species. So there you have it. You're now equipped to transform your data using Pandas powerful group by function.





### **Reshaping dataframes (pivot, stack)**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] After you've got your data, and you've began the data prep phase of your work, there's often a hurdle you've got to jump before you can really extract inside. Often your data simply is not in the appropriate format, or level of granularity needed to begin creating those snazzy visualizations or machine learning models. Luckily, pandas offer several easy to use and flexible functions for you to reshape your data. The first of these is called pivot. This allows you to take away a level of granularity from your rows and pivot that to your columns. You simply identify the index you want to use, the variable you want to pivot from your rows to your columns, in which values you want to populate your data frame. Note, pivot doesn't perform any kind of aggregation. So for this to work, the index and columns you choose, will have to be unique combinations. To show this, we'll create the following data frame. Notice there's two teams per region, and revenue and cost data for each region and team. Great, now let's pivot team to columns, using region as our index. For values, we'll use revenue. Now you see, region is our index and teams is now up top. The values here represent revenue. Next, we'll use stack to work in the opposite direction. That is, we'll pivot column labels to our rows. But first we need to set a multi-index for our data frame. To do this, we use set index and specify region and team. All right, now I'll call the stack function and create a new data frame called stacked. Excellent, so now you can see individual rows for revenue and cost and the values for both now lives within the same column. Similarly, we can reverse this process with unstack, that is we'll pivot row labels back to our columns. And now you see revenue and costs each have their own column once again. Note that this is simply the default behavior and you can actually specify at what level in your multi-index you want to unstack. The default is negative one or the inner most index, and we could change that to negative two or negative three, to unstack by team or region respectively, or better yet you can specifically name the index you want to unstack. For example, let's specify region. Great, now we've got region up top. One other way to reshape your data frames to be mindful of is melt. This allows you to specify ID variables from your columns which will stay intact. All remaining columns or measure variables will be pivoted from the column to the row level. Returning to our original data frame, we're going to designate region and team as our ID variables, then pivot revenue and cost back to the row level. Great, now all of the reshaping approaches so far require that we have unique combinations of the row and column variables that we're pivoting if that's not the case in your data. So for example, say we had multiple revenue entries for each region and team, you'd want to use the handy pivot table function. Here, we set an index, determine which variables we want pivoted to the columns if any, then designate our values. We'll go with revenue here. By default, pivot table uses mean to aggregate, but you can specify the method you want to use as well. Excellent, the average revenue by region is higher for team one than for team two. Now, one last thing, let's add region to columns to see how that changes it. Now we have the average for each region and team. Basically, pivot table acts like a pandas group by, then unstacks by the variable you specify in columns. Between these methods, there is always a way to get your data in the format you need to take it to the next level. When in doubt, PivotTable is the most flexible of the bunch and is a great place to start.

### **Merging (merge, join) and concatenating (concat) dataframes**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Presenter] Soon after you've gone through the steps to clean up your data frame, you may find it's time to incorporate other data alongside it. Here, we'll discuss the many options for emerging and combining data from different data frames. Let's dive in. We'll start by creating two different data frames consisting of letters and numbers. Note the overlap between these two. If you're familiar with SQL, you've heard of joins specifically left joins versus inter-joins versus right joins. We'll start by performing a left join between our two data frames. And the function to use in Pandas is called merge. To merge two data frames, start with your first data frame aka the left data frame then call merge and pass in the second data frame aka the right data frame. You'll also want to specify the join type. In this case a left join and the join criteria. Here will match on the number column in both data frames. Excellent, so you see DF1 shows up in its entirety and only those letters in DF2 that had numbers matching into DF1 came through. That's the definition of a left join. Note the underscore x and underscore y suffixes that have been added as a result of emerge. We'll address that next. The next type of join to use is an inner join. Here will only capture the overlap between our two data frames. Often the column you'll be joining on will have different names in your two data sources. In this example, we use left on and right on to specify the join criteria for each of our data frames. Note, you can join on multiple columns here as well. Great, we see two rows where the numbers match into both data frames. We can also perform what's called a right join which essentially reverses the order of the left join. Also, we're going to use the suffixes argument to specify how we want our columns to look after joining. There you go. Note that all data from DF2, our right data frame was returned along with those that matched from DF1. Also see the underscore right suffix applied to the letter column from DF2. Another cool operation in Pandas is to union your data frames. Essentially stacking them on top of each other. We'll use pd.concat to do this. When using pd.concat, pass the list of the data frames you want to combine and be sure to reset the index and specify drop equals true. Now we have eight rows of data from both data frames, one on top of the other. Note, the indexes have been reset. You'll notice that D and C are repeated in our dataset. And often we want to ensure uniqueness when unioning data frames. To do this, we can rerun and specify drop duplicates. This drop two rows and now we have uniqueness in our data frame. You can also use pd.concat to stack data frames horizontally by specifying access equals one which refers to our column access. The append function is another handy way to stank data frames vertically. Here we'll append a series as a new road to our data frame. When creating the series, I'll pass the columns from DF3 into the index to simplify the append process. There you go. Now we've got one more row at the bottom with letter Z in number 26. The last topic we'll cover is joining data frames using the index values. To do this, use the join function. We'll create another data frame then join that to our DF2. We don't need to specify the join criteria since it will be matching by index but we do need to add in a suffix for repeated column names. Fantastic, we've run through the most fundamental ways you can join data frames. When you've mastered these approaches, you'll find just about anything is possible with Pandas.

### **Mapping variables into groups**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Sure enough, pandas has loads of capabilities to create categories, labels and buckets within your datasets, all with the goal of deriving insights from your qualitative data. Let's investigate a few options. First, we'll create a data frame for counts of salmon on the Skokomish river in Washington state. Here we have species of salmon, where the population is from and they count recorded for those fish. One really handy function in Pandas is called cut. This allows you to create bins for numerical data, then apply those to your data frame. We're going to create bins of 2000 for our count data. Notice here, I important NumPy to make use of NumPy dot inf or infinity, for the top end of our buckets. We label each of the buckets as well, from low return to high return. Next, we create a new column in our data frame called count category. And we pass the count column along with the bins and the labels we've created. This creates our buckets. Great, now you can see the bucketing was successful. Another useful tactic is to create a dictionary which maps values in your data frame into groupings. Here, we'll create a dictionary mapping species to its federal endangered species status. Next, we'll apply the mappings to our species column using map. It looks good. Pandas even has a specific categorical data type that's useful for us as well. We're going to convert our count category column using Pandas dot categorical. Notice, we pass the column of our data frame, ordered equals true which will make this variable ordinal meaning high return has a higher order, and pass our labels from above to categories. In the printout, you can see this hierarchy taking place, low returns less than below average return, and so on. In fact, we can even use this ordering to sort our data. To do this, we'll use sort values by the count category column, and it'll be descending. Look at that, the data is now sorted from the highest return category to the lowest. Another function, which is especially useful for statistics and machine learning is get\_dummies. This allows us to take a categorical variable and perform what's called one hot encoding. By which we create a column for each category and place a one, if that is true, or zero, if not. Here's an example of how that looks. So there you have it, go forth and categorize your data.



### **Plotting with pandas**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Pandas is not a data visualization library by any stretch of the imagination. However, it does have built-in functionality to make use of Matplotlib on the fly. This is especially useful when performing exploratory data analysis, and you want a quick visual representation of your data. As a general rule, if you're presenting your findings with non-technical audiences, there are better options for crafting your visuals. To start us off, we're going to create a time series data frame with two randomly-generated values per period. There you go. Now, it couldn't be simpler for us to visualize this data in Pandas. Simply call plot. Now, we see a graph with two lines spanning the 50 periods in our dataset. An auto-generated legend has been placed on our visual as well. Notice the use of the semi-colon at the end of our function. This is included to prevent messy code output from being printed out along with our graph. Now, let's review a few more options for visuals. Here, we'll generate an area graph which is similar to our line chart above, but it's filled. Note, we passed stacked equals false, but a stacked area vis is an option as well. Now, you can see our lines have actually been stacked one upon the other. Next, let's import the iris dataset. We can see histograms for our dataset very simply by calling .hist on our data frame. Here, the output shows a histogram for each variable in the iris dataset. This is the type of visual that can really speed along your process by calling out trends in a clean visual manner. Now, by instead calling plot.hist, we can overlay the histograms of our variables on the same axis. Note, when you overlay visuals, you'll want to increase the transparency by altering the alpha parameter. Here, alpha is set to 0.5. The next visualization we'll generate is a scatter plot. We'll designate sepal width for the x-axis and sepal length for the y-axis. We can then plot each iris observation and assign a color corresponding to the species. Notice here, we use .map to translate each species to a color for our plot. This is another insightful visual. We see pretty definitive clusters forming between our three species. The last plot type for us to review is called the scatter matrix. It will generate scatter plots showing the relationship between each of your variables, and along the diagonal, you will see histograms showing how your observations are distributed. Notice the figsize in our scatter matrix. This parameter allows you to better fit your visuals to the space available on your screen. Reviewing our visual, we can see that petal length and petal width have a very strong positive correlation. This histogram is simply showing the distribution of observations for petal width. Overall, this is a very nice visual to get a very quick understanding of your data. While Pandas' visuals may not be the snazziest you'll find, their simplicity and ease of use are hard to beat when you need insights on the fly. Next time you're performing exploratory data analysis, try some of these features out.

### **Correlations and statistical functions**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Long before machine learning and AI became the buzzwords of the century, the humble discipline of statistics established the foundation for drawing inference and insight from our data. With pandas, you have several statistical functions right at your fingertips. Before we explore the statistics capabilities of pandas, let's import the iris dataset. First, let's explore some of the built-in measures for central tendency. By simply calling .mean on our data frame, pandas computes the average for each of our numeric variables. We can similarly return the median values with .median. Now, if we want to return the most frequent observations for each of our variables, we can utilize the mode function. This output takes a moment to interpret. Knowing that a data set can have multiple modes per variable in the case of a tie, we see that our output returns the mode at index zero, but in the case of a tie, it will add additional rows to represent that. In the iris dataset, there is a three way tie for species as there are 50 records apiece, so all three show up. We also have a tie for frequency in petal length, so you see two values there as well. Measures of central tendency are valuable, but without also knowing how your data is distributed, they can be misleading. With pandas, you can easily return the standard deviation for variables in your data frame. This will tell you the extent to which these variables deviate from their mean. There you go. These are the standard deviations for all our measures. It's often even more helpful to have a visual representation of how your data is distributed. That's where the box plot visual comes in handy. You can easily produce this visual by calling .boxplot on your data frame. There it is. In this visual, the rectangles represent the extent from the first quartile to the third quartile in your data. Remember, one quartile represents one fourth of your distribution. The whiskers extending beyond are meant to show the range of your data. Dots exceeding the whiskers are considered outliers. Now, the quickest way to get an understanding of your data is the describe function. Simply call this on your data frame to yield a thorough list of descriptive statistics. Many of the calculations we performed above are provided and quartile values show up as well. For more advanced analysis in statistical modeling, it becomes a really important to understand how variables in your data set interact with each other, specifically, how well correlated they are to each other. Correlation simply shows the extent and direction in which two variables are related. You can see pairwise correlations between all your variables with .corr. Our output allows you to see the correlation between any two variables. A positive number demonstrates that as one variable increases, the other tends to as well, while negative shows the opposite. The closer to one or negative one, the stronger the relationship. Note the diagonal is all ones as we're correlating variables with themselves. The usefulness of this output can't be overstated. It's referred to as the correlation matrix and is often a good first step before proceeding to machine learning or statistical modeling. To make this even easier to interpret, we can create a heat map from this matrix. In pandas, calling style.background\_gradient on our correlation matrix yields this snazzy visual. Now with color, it's very clear how strongly and in which direction our variables are correlated. Pandas has really made its reputation as an all around one stop shop for data analysis. And with access to these functions, you can probably see why. In your next project, incorporate these statistical techniques and you'll be well rewarded.

### **Accelerate EDA with pandas-Profiling**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] The popularity and usefulness of Pandas has put it at the center of the Python data analysis universe. So it's no surprise that it's spawned countless sister packages and dependencies throughout the Python ecosystem. One related package, which I highly encourage you to use, particularly if you want to accelerate your exploratory data analysis, is Pandas Profiling. This simple and intuitive package will effectively take your data frame and output an interactive and highly comprehensive profile for each variable in your data. It surfaces the descriptive statistics for each variable, provides a correlation matrix, and assesses the data quality, and more. Much of the proceeding work in this course can more readily be done with Pandas Profiling. Excited? Me too. Let's dive in. In addition to importing Pandas, we'll want to import ProfileReport from pandas\_profiling. If you don't have this installed, you can simply install via pip. Let's profile our iris dataset. To create a profile, use the ProfileReport function, pass in your data frame, and provide it with the title. Next, we'll output the profile in our notebook. Notice the handy progress bar updating. Okay, now let's investigate. To start, we get an overview of our data, the number of observations, how many variables there are, whether there's null values, duplicate rows, everything. You'll notice there's even warnings that are surfaced, which you can see by clicking here. These warnings may or may not be a problem for you, but nonetheless, they're good to be mindful of. We can see profiles of each of our variables in the Variables section. Here, you see some descriptive statistics for sepal\_length. There's also a histogram of values to the right. We can toggle details to see even more detail. Here, we see very in-depth descriptive statistics. You can even see a histogram, common values, and the extreme values in your dataset. This continues for each variable. We can also see interactions between variables. Below, you'll see a scatter plot of the two variables of your choosing. This plot shows all the values of sepal\_width and sepal\_length. This allows you to eyeball any correlations or relationships that you think are meaningful. Speaking of, we can also output a correlation matrix. This simple visual can tell you whether your variables are correlated to one another and the direction of that correlation. Blue is a positive correlation and red is a negative correlation. The darker the color, the stronger the relationship. Another handy feature is to see your missing values. Thankfully, we're not missing any in this dataset. At the tail end here, you also see a head of your data frame and a tail. Lastly, if you have duplicate rows, you'll find them here. Our dataset has one duplicate. After reviewing your profile, you can output this as an easily sharable HTML file. To do this, use the to\_file function and name your HTML file. Using Pandas Profiling has become second nature for me, and to get the most out of Pandas, I highly recommend you incorporate it into your projects as well.

**Pandas Profilling —VVVI VVVVIVIVIVIVIVIVIVIVIVIVIIVIVIVII**

### **Accelerate EDA with pandas-Profiling**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] The popularity and usefulness of Pandas has put it at the center of the Python data analysis universe. So it's no surprise that it's spawned countless sister packages and dependencies throughout the Python ecosystem. One related package, which I highly encourage you to use, particularly if you want to accelerate your exploratory data analysis, is Pandas Profiling. This simple and intuitive package will effectively take your data frame and output an interactive and highly comprehensive profile for each variable in your data. It surfaces the descriptive statistics for each variable, provides a correlation matrix, and assesses the data quality, and more. Much of the proceeding work in this course can more readily be done with Pandas Profiling. Excited? Me too. Let's dive in. In addition to importing Pandas, we'll want to import ProfileReport from pandas\_profiling. If you don't have this installed, you can simply install via pip. Let's profile our iris dataset. To create a profile, use the ProfileReport function, pass in your data frame, and provide it with the title. Next, we'll output the profile in our notebook. Notice the handy progress bar updating. Okay, now let's investigate. To start, we get an overview of our data, the number of observations, how many variables there are, whether there's null values, duplicate rows, everything. You'll notice there's even warnings that are surfaced, which you can see by clicking here. These warnings may or may not be a problem for you, but nonetheless, they're good to be mindful of. We can see profiles of each of our variables in the Variables section. Here, you see some descriptive statistics for sepal\_length. There's also a histogram of values to the right. We can toggle details to see even more detail. Here, we see very in-depth descriptive statistics. You can even see a histogram, common values, and the extreme values in your dataset. This continues for each variable. We can also see interactions between variables. Below, you'll see a scatter plot of the two variables of your choosing. This plot shows all the values of sepal\_width and sepal\_length. This allows you to eyeball any correlations or relationships that you think are meaningful. Speaking of, we can also output a correlation matrix. This simple visual can tell you whether your variables are correlated to one another and the direction of that correlation. Blue is a positive correlation and red is a negative correlation. The darker the color, the stronger the relationship. Another handy feature is to see your missing values. Thankfully, we're not missing any in this dataset. At the tail end here, you also see a head of your data frame and a tail. Lastly, if you have duplicate rows, you'll find them here. Our dataset has one duplicate. After reviewing your profile, you can output this as an easily sharable HTML file. To do this, use the to\_file function and name your HTML file. Using Pandas Profiling has become second nature for me, and to get the most out of Pandas, I highly recommend you incorporate it into your projects as well.

**Geo PAndas →?>>>>>>>>VVIVVVIVVVIVVVI**

### **Beyond pandas with Dask and Koalas (Spark)**

Selecting transcript lines in this section will navigate to timestamp in the video

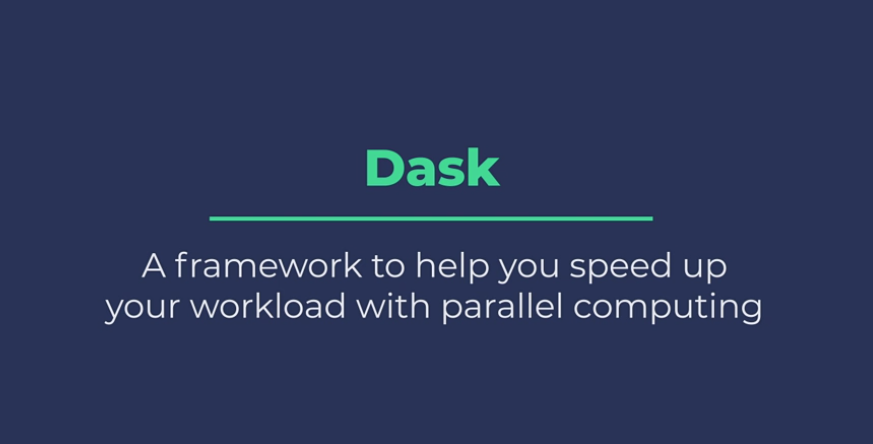
- [Instructor] There may come a time when the volume of your data has become so large, that you find using Pandas to be constraining. Thankfully there have been rapid advancements in big data processing. And in this lesson, we'll discuss two frameworks. Dask and Spark, that have proven to be especially useful. We'll also see how Pandas can be easily translated into these cutting edge approaches. So what is Dask? At a high level, Dask is a framework to speed up your Python workload using parallel computing. So instead of running all tasks sequentially, the Dask scheduler allows for simultaneous computing. This takes full advantage of all the compute you have available. One of the key benefits of Dask, is it is very compatible with all the existing work you do with Python and Pandas in particular. Dask has a dataframe concept, just as Pandas does. In fact you can think of a Dask dataframe as consisting of a series of Pandas dataframes. Let's import our iris dataset as a Dask dataframe using the read\_csv function, which should look pretty familiar to us. Generally speaking, this dataframe will behave just like we're used to. The next framework to learn about is Apache Spark. Spark is written in a language called Scala, and is easily called in Python using what's called PySpark. It's also optimized for big data, and distributes computing across clusters of compute for fast data processing. To aid in the transition from Pandas, there's a package cleverly named Koalas. Which is the Pandas API on top of Apache Spark. This is the standard block to import Koalas, which we'll import as ks. Now let's create a Pandas dataframe. Here we have a randomly generated dataframe with four columns and five rows. To transition a Panda's dataframe into Koalas, you can use the from\_pandas function. Sure enough, it looks and acts pretty similarly. But now you can take advantage of the distributed computing powers of Spark. Dask and Spark are each entire courses unto themselves. So if you need to scale up your efforts to process big data, this is where you should look. You can easily get started with Dask on your local machine. And to get a fully managed Spark experience, you can sign up for the free community edition of Databricks.

**DASK and SPARK and kaolas**

### **Your path forward using advanced pandas functions**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Brett] Congratulations, you've worked your way through advanced pandas and we've covered a lot of material. As you continue to work with pandas, you'll find as I have that there's always something more to learn and to try out for yourself. One way to hone your panda skills is to have a textbook to work through in reference, Wes McKinney's Python for Data Analysis is an excellent resource. And please feel free to reach out to me, Brett Vanderblock and I'm happy to help you along the way. Once you feel you've mastered pandas rest assured there's plenty more still to learn. For starters, you can work with geospatial data and geo pandas, and definitely be sure to experiment with big data frameworks, such as DAS and Spark. I hope you've enjoyed this course and happy coding.



****